Submitted to Journal of Cryptology, August 2000.

A SECURE PUBLIC-KEY SIGNATURE SYSTEM WITH EXTREMELY FAST VERIFICATION

DANIEL J. BERNSTEIN

ABSTRACT. This paper presents a variant of the Rabin-Williams public-key signature system. The new system offers the same security and signing speed but much faster verification. Generic attacks against this system are provably as difficult as factorization.

1. Introduction

This paper establishes a new speed record for signature verification in RSA-type public-key signature systems.

The signature system introduced here is a variant of the Rabin-Williams system, which held the speed record for two decades. The systems have the same key size, security, and signing speed.

A signature in this system takes more space than a Rabin-Williams signature: it is, in fact, a Rabin-Williams signature with some extra information attached. A receiver can discard the extra information to save space, and regenerate the extra information later. See Figure 1.

In each of these systems, the public key is an integer. An attacker can break the system if he can factor the integer into primes. This paper restricts attention to 1536-bit public keys, i.e., integers between 2^{1535} and 2^{1536}. If it ever becomes practical to factor keys of this size, users will have to move to larger keys, making appropriate modifications in the details of the system.

This system, like the Rabin-Williams system, has the best security guarantee known for any RSA-type system: if an attacker can quickly forge signatures under an arbitrary hash function, then the attacker can quickly factor public keys. See section 6. Beware, however, that many specific hash functions are breakable.

2. Keys

A **secret key** is a pair of prime numbers (p, q) such that $p \bmod 40 = 3$, $q \bmod 40 = 7$, $4 \cdot 2^{765} < p < 5 \cdot 2^{765}$, and $2^{800}\overline{\pi} < pq < 2^{800}(\overline{\pi} + 1)$. Here $\overline{\pi} \approx 2.16 \cdot 10^{221}$ is the integer obtained by writing the first 222 digits of π in reverse order; in hexadecimal it is

```
         9949AB0B 00D505ED 84F87BF4 6E10D628 5C86E943 7729912B DDD99955
B3877272 B6D65463 1772C6BE F62755DC 39DBB3A3 47D512C4 71838883 72BE4B91
59883460 B03ECBF4 88DAFE8B 6E49454C 010C23FE C6F682E4 9E241C1A 28E0B6F5.
```

Date: 20000809.

1991 *Mathematics Subject Classification.* Primary 94A60; Secondary 11Y16.

The author was supported by the National Science Foundation under grant DMS–9600083.

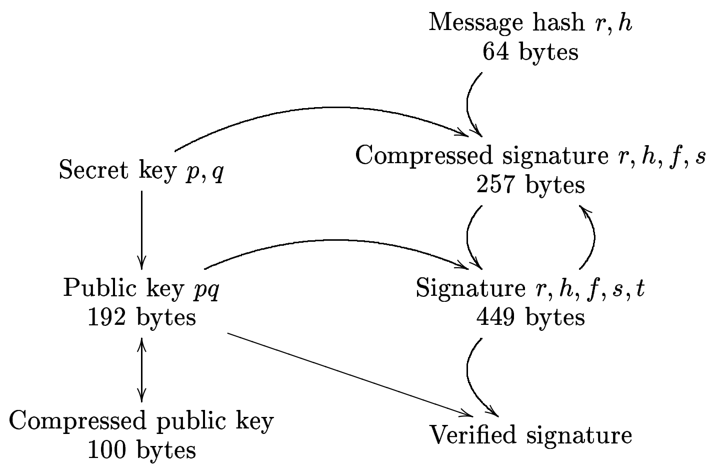

FIGURE 1. Data flow in the signature system.

The following procedure seems to generate a random secret key with a nearly uniform distribution. Generate a uniform random number p between $4 \cdot 2^{765}$ and $5 \cdot 2^{765}$ satisfying $p \bmod 40 = 3$; repeat until p is prime. Generate a uniform random number q between $\lceil 2^{800}\overline{\pi}/p \rceil$ and $\lfloor 2^{800}(\overline{\pi}+1)/p \rfloor$ satisfying $q \bmod 40 = 7$; repeat until q is prime.

I check whether p is prime by the following combination of tests:

- Compute $p \bmod r$ for all primes $r < 100$. If $p \bmod r = 0$ then p is composite.
- Compute $2^{(p-1)/2} \bmod p$. If the result is anything other than 1 or $p-1$ then p is composite.
- Find the first integer $b \geq 3$ such that the Jacobi symbol of $b^2 - 4$ modulo p is -1. In the ring $(\mathbf{Z}/p)[x]/(x^2 - bx + 1)$ compute $x^{(p+1)/2}$. If the result is anything other than 1 or $p-1$ then p is composite. (In this application, b is always 3, since $p \bmod 5 \in \{2, 3\}$.)

There is no known example of a composite number p that slips past all of these tests; I am confident that nobody will ever find one by accident. For motivation see, e.g., [18, section 10]. See [6, chapter 9] and [17] for some practical methods to *prove* that p is prime. Beware that the techniques of [17] do not apply to q.

Public keys. The **public key** corresponding to (p, q) is the product pq. This product is between 2^{1535} and 2^{1536}.

There is no known practical algorithm that will compute $pq \mapsto (p, q)$ with non-negligible probability. The number field sieve needs roughly 2^{100} operations to find p and q; see [13]. Presumably the attacker does not have this much time, so he will instead use Lenstra's elliptic curve method, which has a tiny chance of discovering p and q after a small amount of computation; see [14]. If he can afford 2^{80} such computations then his chance of success is still below 2^{-40}.

Compressed public keys. One can recover pq from $pq \bmod 2^{800}$ as follows: $pq = 2^{800}\overline{\pi} + (pq \bmod 2^{800})$. Consequently one can store and transmit pq in only 100 bytes.

The idea of prespecifying some bits of public keys to save space was patented by Vanstone and Zuccherato in 1994. Fortunately, the patent is invalid, because the idea had been published years before. See, e.g., [10, page 467].

One might object to the requirement that $\overline{\pi}$ be used in every public key: there could be an exceptionally fast factorization algorithm for numbers containing that bit pattern. This objection is fundamentally flawed. Any particular number pq is instantaneously factored by the algorithm "print p and q"—but how is an attacker supposed to *find* that algorithm? Church's thesis states that the attacker's behavior is simulated by some predetermined probabilistic algorithm A; the crucial question is whether *that* algorithm succeeds against a randomly generated pair (p, q). Define α as the probability that A succeeds against a pair (p, q) with $2^{1535} < pq < 2^{1536}$, and define $\beta(t)$ for $2^{735} < t < 2^{736}$ as the conditional probability that A succeeds when $\lfloor pq/2^{800} \rfloor = t$. Then α is the average of $\beta(t)$, weighted by the distribution of t, which is nearly uniform. Consequently there cannot be a noticeable fraction of t's for which $\beta(t)$ is much larger than α. It would be a cosmic conspiracy beyond belief for the choice $t = \overline{\pi}$ to be especially bad.

3. Signatures

The signature system is parametrized by (1) a set X of **messages** and (2) a **hash function** $H : \{r \in \mathbf{Z} : 0 \le r < 2^{256}\} \times X \to \{h \in \mathbf{Z} : 0 < h < 2^{1531}, h \bmod 8 = 1\}$.

Let n be a public key. A **signature of** m **under** n is a vector of integers (r, h, f, s, t) such that $0 \le r < 2^{256}$, $h = H(r, m)$, $f \in \{-2, -1, 1, 2\}$, $0 \le s < 2^{1536}$, $0 \le t < 2^{1536}$, and $s^2 = tn + fh$.

Each message has many signatures. Signers are required to follow certain rules in generating signatures; some signatures must not be generated even though they will be accepted by verifiers. See section 4 for the complete signing procedure and section 6 for a security analysis.

Verifying signatures. Let m be a message, and let r, h, f, s, t be integers with $0 \le r < 2^{256}$, $0 \le h < 2^{1531}$, $f \in \{-2, -1, 1, 2\}$, $0 \le s < 2^{1536}$, and $0 \le t < 2^{1536}$. Define $c = s^2 - tn - fh$. Checking that (r, h, f, s, t) is a signature of m under n means checking that $h = H(r, m)$ and that $c = 0$.

One way to check whether $c = 0$ is to check whether c is divisible by a small secret prime. Consider, for example, prime numbers between 2^{114} and 2^{115} congruent to 2 modulo 5; there are more than $40 \cdot 2^{100}$ such primes. If $c \ne 0$ then c cannot be divisible by more than 26 of these primes, since $|c| < 2^{3072}$; so the chance that a uniform random prime will divide c is below 2^{-100}. This possibility can be safely ignored.

So choose a prime u; compute $s' = s \bmod u$, $t' = t \bmod u$, $n' = n \bmod u$, and $h' = h \bmod u$; finally compute $((s')^2 - t'n' - fh') \bmod u$. The result is $c \bmod u$: it is zero if $c = 0$, and it is practically guaranteed to be nonzero if $c \ne 0$.

Note that the same prime u can be used to check many signatures. If part of h is determined by the hash function H then that part can be precomputed mod u. Similarly, n' can be saved if there are several signatures to check under the same public key n.

It may be inconvenient for the verifier to store a secret random prime. An alternative is to simply compute c and verify that $c = 0$; or to verify that c is divisible by a set of moduli whose least common multiple exceeds a computed bound on $|c|$. The moduli can be selected to support fast division.

Software to perform these computations, using the techniques explained in [5], is available from `http://cr.yp.to/sigs.html`.

Compressed signatures. A **compressed signature of m under n** is a vector (r, h, f, s) such that (r, h, f, s, t) is a signature of m under n for some t.

Let m be a message, and let r, h, f, s be integers with $0 \le r < 2^{256}$, $0 \le h < 2^{1531}$, $f \in \{-2, -1, 1, 2\}$, and $0 \le s < 2^{1536}$. There are several good ways to compute an integer t such that $s^2 - fh = tn$ if and only if $s^2 - fh$ is divisible by n. Then (r, h, f, s) is a compressed signature of m under n if and only if (r, h, f, s, t) is a signature of m under n.

Signatures can be compressed further: one can recover (r, h, f, s) from (r, s). Bellare and Rogaway in [4, section 6] suggested a different method of signature randomization in which signatures can be compressed to simply s; but their method is incompatible with fast verification.

A typical hash function. In my software, messages are strings of bytes, and H is defined as follows. Feed (r, m) through Merkle's Snefru-8/256 (see [16]), little-endian, to obtain an integer z with $0 \le z < 2^{256}$. Then $H(r, m) = 2^{320}\overline{e} + 2^{64}z + 1$. Here $\overline{e} \approx 9.07 \cdot 10^{363}$ is the integer obtained by writing the first 364 digits of $\exp 1$ in reverse order; in hexadecimal it is

```
                            20EC2CF 9A875185 23FE3A9D 2F0146A5 91B19A5B 1D9B5799
AB577BA7 D2CF9561 A90606F1 48BC9CDB 98BFBAD3 27663C51 56C5EC8F DAF79AB5
7BC4FA99 5083FB9D EEF66C84 10F54365 DA041484 817305DD 16A6294E 28980AD7
C3D172C9 D0454CAD A219FC7B A0A5E47F B16FDAD7 7A4E33A3 9E673D5B F932E1F1
210405D4 CC46B9E3 EEDFDA4E 043D1196 81144131 4ECE679B 97D28C87 6467D34C.
```

One can compress $H(r, m)$ to the 32-byte value z. Figure 1 assumes implicitly that this is done.

Notes. Rivest, Shamir, and Adleman in [21] proposed the signature equation $s^k \equiv h \pmod{pq}$, with public key (pq, k); here k is a randomly chosen integer coprime to $(p-1)(q-1)$. Knuth in [12, page 388] proposed the much more easily verified equation $s^3 \equiv h \pmod{pq}$. Rabin in [20] proposed $s^2 \equiv h \pmod{pq}$. But what if h is not a square modulo pq? Rabin suggested trying a new r. Williams in [22] suggested the answer used above: require $p \bmod 8 = 3$ and $q \bmod 8 = 7$, and then allow $s^2 \equiv fh \pmod{pq}$ with $f \in \{-2, -1, 1, 2\}$.

In March 1997 on `sci.crypt` I suggested providing $t = (s^2 - fh)/pq$ as part of the signature. This makes verification much easier with no effect on security. The new signature equation $s^2 = tn + fh$ is a ring equation; testing a ring equation by mapping it to a random quotient ring is a standard technique, as is proving a ring equation by mapping it to enough quotient rings.

Rabin in [19] pointed out that the choice of hash function H is crucial for security. For example, given n, an attacker must not be able to find (m, r, s) with $s^2 \bmod n = H(r, m)$; otherwise $(r, H(r, m), 1, s, (s^2 - H(r, m))/n)$ is an unauthorized signature of m. The original RSA system was blatantly insecure: the set of messages was $\{0, 1, \ldots, pq - 1\}$, and $H(r, m)$ was simply m.

See [11, page 13] and [10] for more examples of hash functions. Some hash functions allow compression of m ("message recovery") given h.

4. Standard signatures

Let m be a message, and let n be a public key. A **standard signature of** m **under** n is a vector of integers (r, h, f, s, t) such that

- $0 \leq r < 2^{256}$,
- $h = H(r, m)$,
- $s^2 = tn + fh$,
- $n/2 < s \leq n$,
- either s or $-s$ (not both!) is a square modulo n, and
- f is the first integer in the sequence $1, 2, -1, -2$ such that fh is a square modulo n.

Observe that if (r, h, f, s, t) is a standard signature of m under n then it is also a signature of m under n.

The security analysis in section 6 assumes that (1) the signer always generates standard signatures and (2) the signer chooses a uniform random r for each message to be signed. Algorithm S achieves these results.

The requirement to generate standard signatures must not be violated under any circumstances. No guarantees are provided for signers who generate nonstandard signatures.

In contrast, non-uniform choices of r are not a disaster; even if r is always chosen as 0, one can combine my proof in section 6 with the Bellare-Rogaway proof in [4, section 3] to prove a security guarantee similar to Theorem 6.1. However, the guarantee in Theorem 6.1 is quantitatively stronger. The advice to randomize r is due to Rabin in [20].

Algorithm S. Given a message m, a secret key (p, q), the public key $n = pq$, and integers x, y with $xp + yq = 1$, to compute a signature of m under n:

```
1. Choose a uniform random $r \in \{0, 1, \ldots, 2^{256} - 1\}$.
2. Compute $h \leftarrow H(r, m)$.
3. If the Legendre symbol of $h$ modulo $q$ is $-1$, set $e \leftarrow -1$; otherwise set $e \leftarrow 1$.
4. If the Legendre symbol of $eh$ modulo $p$ is $-1$, set $f \leftarrow 2e$; otherwise set $f \leftarrow e$.
5. (Now $fh$ is a square modulo $n$.) Set $a \leftarrow fh$.
6. Compute $c \leftarrow yq(a^{(p+1)/4} \bmod p) + xp(a^{(q+1)/4} \bmod q) \bmod n$.
7. If $c > n/2$, set $s \leftarrow c$; otherwise set $s \leftarrow n - c$. (Now $s^2 \equiv a \pmod{n}$, and
   $n/2 < s \leq n$.)
8. Compute $t \leftarrow (s^2 - a)/n$.
9. Print $(h, f, s, t)$.
```

Theorem 4.1. *Algorithm S prints a standard signature of m under n.*

Proof. By the definition of secret key, $p \bmod 8 = 3$ and $q \bmod 8 = 7$. Conclusions: -1 is not a square modulo q; 2 is a square modulo q; 2 is not a square modulo p.

If $e = 1$ then h is a square modulo q so $2h$ is a square modulo q. If $f = 1$ then h is a square modulo p, hence modulo n. If $f = 2$ then h is a nonsquare modulo p, hence modulo n; but $2h$ is a square modulo p, hence modulo n.

If $e = -1$ then h and $2h$ are nonsquares modulo q, hence modulo n. On the other hand, $-h$ and $-2h$ are squares modulo q. If $f = -1$ then $-h$ is a square modulo p, hence modulo n. If $f = -2$ then $-h$ is a nonsquare modulo p, hence modulo n; but $-2h$ is a square modulo p, hence modulo n.

Therefore $a = fh$ is a square modulo n. Select a square root b.

By construction $c^2 \equiv a \pmod p$. Indeed, $xp \equiv 0$, and $yq = 1 - xp \equiv 1$, so $c \equiv a^{(p+1)/4} \equiv b^{(p+1)/2}$; thus $c^2 \equiv b^{p+1} \equiv b^2 \equiv a$.

By the same argument $c^2 \equiv a \pmod q$; so $c^2 \equiv a \pmod n$. Thus $s^2 \equiv a \pmod n$, so t is an integer, and $s^2 = tn + a = tn + fh$. By construction $0 \le c < n$, so $n/2 \le s \le n$.

Finally, c is a square modulo p and modulo q, hence modulo n. Note that h is not divisible by n, so $c \not\equiv -c \pmod n$. If $c \not\equiv -c \pmod p$ then $-c$ is a nonsquare modulo p, hence modulo n; otherwise $c \not\equiv -c \pmod q$ so $-c$ is a nonsquare modulo q, hence modulo n. □

5. Squares

This section is a prelude to the security analysis in section 6. It shows how an attacker can carry out various tasks involving a public key n, without being given the factorization of n.

A **hash value** is an integer h with $0 < h < 2^{1531}$ and $h \bmod 8 = 1$. A **scribble under** n is a vector of integers (h, f, s, t) such that

- h is a hash value,
- $s^2 = tn + fh$,
- $n/2 < s \le n$,
- either s or $-s$ is a square modulo n, and
- f is the first integer in the sequence $1, 2, -1, -2$ such that fh is a square modulo n.

Thus (r, h, f, s, t) is a standard signature of m under n if and only if $0 \le r < 2^{256}$, $h = H(r, m)$, and (h, f, s, t) is a scribble under n.

Algorithm V generates a uniform random hash value h. At the same time it generates (f, s, t) such that (h, f, s, t) is a scribble modulo n.

Algorithm W generates a uniform random hash value h. At the same time it generates auxiliary information so that a signature involving h determines a square root modulo n of a given number y.

Algorithm H. Given a public key n and a square a modulo n with $0 \le a < n$:

1. Compute an integer z such that $2z \equiv a \pmod n$.
2. Find $f \in \{-2, -1, 1, 2\}$ such that $(2/f)z \bmod n$ is a hash value. If no such f exists, proclaim failure and stop.
3. Set $h \leftarrow (2/f)z \bmod n$. (Now fh is a square modulo n.)
4. If $\gcd\{h, n\} = 1$, go to step 6.
5. (Since h is a hash value, it is not divisible by n; so it has a nontrivial factor in common with n.) Use h to factor n. Check, as in Algorithm S, whether f is the first integer in the sequence $1, 2, -1, -2$ such that fh is a square modulo n. If not, proclaim failure and stop.
6. Print (h, f).

Theorem 5.1. *Algorithm H prints (h, f) if and only if (1) h is a hash value; (2) f is the first integer in the sequence $1, 2, -1, -2$ such that fh is a square modulo n; and (3) $a = fh \bmod n$.*

Proof. Say Algorithm H prints (h, f). Step 2 selects f so that h is a hash value in step 3. Also $fh \equiv 2z \equiv a \pmod n$, so fh is a square modulo n. If $\gcd\{h, n\} = 1$ then f is the only integer in $\{1, 2, -1, -2\}$ such that fh is a square modulo n; if

$\gcd\{h,n\} \neq 1$ then step 7 checks whether f is the first integer in $1, 2, -1, -2$ such that fh is a square modulo n.

Conversely, say (h, f) is a pair satisfying the stated conditions. Then f is found in step 2 of Algorithm H, since there cannot be two hash values among $z \bmod n, 2z \bmod n, -z \bmod n, -2z \bmod n$; and h is found in step 3. Algorithm H will not fail in step 5, since f is the first integer in $1, 2, -1, -2$ such that fh is a square modulo n. □

Algorithm U. Given a public key n, to print a uniform random square modulo n:

```
1. Generate a uniform random integer $x \in \{0, 1, \ldots, 2^{1536} - 1\}$.
2. If $x \geq n$, go back to step 1.
3. If $x = 0$, print 0 and stop.
4. Generate a uniform random bit. If it is 0, go back to step 1.
5. If $\gcd\{x, n\} > 1$, print $x^2 \bmod n$ and stop.
6. Generate a uniform random bit. If it is 0, go back to step 1.
7. Print $x^2 \bmod n$ and stop.
```

Theorem 5.2. *Algorithm U prints a uniform random square modulo n.*

Proof. Consider a square c modulo n. If $c \equiv 0$ then c has one square root, namely 0; step 1 sets $x \leftarrow 0$ with probability $1/2^{1536}$, at which point step 3 will print c. If c is nonzero but divisible by p or q then c has two square roots; step 1 sets x to a square root of c with probability $2/2^{1536}$, and step 4 allows x to pass with conditional probability $1/2$, at which point step 5 will print c. If c is coprime to n then c has four square roots; step 1 sets x to a square root of c with probability $4/2^{1536}$, and steps 4 and 6 allow x to pass with conditional probability $1/4$, at which point step 7 will print c. To summarize: Each invocation of step 1 leads to the algorithm printing c with probability $1/2^{1536}$, no matter what c is. □

Algorithm V. Given a public key n, to print a uniform random scribble under n:

```
1. Generate a uniform random square $c$ modulo $n$ by Algorithm U.
2. If $c > n/2$, set $s \leftarrow c$; otherwise set $s \leftarrow n - c$.
3. Compute $(h, f)$ from $s^2 \bmod n$ by Algorithm H. If Algorithm H fails, go back
   to step 1.
4. Compute $t \leftarrow (s^2 - fh)/n$.
5. Print $(h, f, s, t)$.
```

Theorem 5.3. *Algorithm V prints a uniform random scribble under n.*

Proof. By construction $n/2 < s \leq n$. By Theorem 5.1, h is a hash value, f is the first integer in the sequence $1, 2, -1, -2$ such that fh is a square modulo n, and $s^2 \equiv fh \bmod n$; so t is an integer and $s^2 = tn + fh$. Exactly one of s and $-s$ is a square modulo n, as in Theorem 4.1. Thus the output of Algorithm V is a scribble under n.

Now consider any scribble (h', f', s', t'). By definition exactly one of s' and $-s'$ is a square modulo n. I claim that Algorithm V prints (h', f', s', t') if and only if, in step 1, c is that square. Each square modulo n is generated with equal probability, so each scribble is generated with equal probability.

Proof of the claim: If $c \equiv \pm s'$, then s must equal s' in step 2; thus $f'h' \equiv s^2 \pmod{n}$, so $(h, f) = (h', f')$ by Theorem 5.1; and $t = t'$ in step 4. Conversely, if $s = s'$, then certainly $c \equiv \pm s'$. □

Theorem 5.4. *The map $(h, f, s, t) \mapsto h$ is a bijection from scribbles under n to hash values.*

Proof. Injectivity: Consider two scribbles (h, f, s, t) and (h, f', s', t').

By assumption f is the first integer in the sequence $1, 2, -1, -2$ such that fh is a square modulo n, and f' is the first integer in the sequence $1, 2, -1, -2$ such that $f'h$ is a square modulo n, so $f = f'$.

By assumption $s \equiv \pm u^2 \pmod{n}$ for some integer u. Write $a = fh$ and $c = a^{(pq-p-q+5)/8} \bmod n$. Then $u^2 \equiv c \pmod{n}$: indeed, $u^2 \equiv u^{p+1} \equiv a^{(p+1)/4} \equiv c \pmod{p}$, and similarly for q. Hence $s \equiv \pm c \pmod{n}$. If $c > n/2$ then $s = c$ since $n/2 < s < n$; otherwise $s = n - c$ since $n/2 < s < n$. The same logic applies to s', so $s' = s$.

Finally $t'n = s^2 - fh = tn$.

Surjectivity: Given h, construct f, s, t as in Theorem 4.1. □

Algorithm W. Given a public key n, and an integer y coprime to n that is a square modulo n:

1. Generate a uniform random square w modulo n by Algorithm U.
2. Compute (h, f) from $w^2 y \bmod n$ by Algorithm H. If Algorithm H fails, go back to step 1.
3. Print (h, f, w).

Theorem 5.5. *Algorithm W prints (h, f, w) where h is a uniform random hash value, f is the first integer in the sequence $1, 2, -1, -2$ such that fh is a square modulo n, and $fh \equiv w^2 y \pmod{n}$.*

Proof. By Theorem 5.1, h is a hash value, f is the right integer, and $fh \equiv w^2 y \pmod{n}$.

There is a square root x of y modulo n such that x is a square modulo n. Write $c = wx \bmod n$. Then c is a uniform random square modulo n, with $c^2 \bmod n = w^2 y \bmod n$, so the distribution of (h, f) in step 2 of Algorithm W is the same as the distribution of (h, f) in step 3 of Algorithm V. By Theorem 5.3, h is uniformly distributed. □

Notes. Algorithm V is reasonably fast. There are 2^{1528} hash values, and therefore 2^{1528} choices of c that succeed in step 1, out of about $n/4$ squares modulo n. Thus step 1 is repeated about $n/2^{1530} < 64$ times on average. Similar comments apply to Algorithm W.

In practice it is safe to assume that a uniform random 1536-bit number will not have any factors in common with n. One can therefore skip steps 4 and 5 of Algorithm H and steps 3 through 6 of Algorithm U.

6. Security

The point of this section is that, unless H has some exploitable structure, the attacker's only hope is to factor the public key n. More precisely: if there is an efficient *generic* algorithm to forge signatures under n—i.e., an algorithm that works with most hash functions H—then there is an efficient algorithm to compute a square root of any given square modulo n.

The threat model considered here is an **adaptive chosen-message attack**. The attacker supplies a message to the user; the user publishes a standard signature of the message. The attacker repeats this process any number of times. He then attempts to sign a *different* message.

Generic attacks. A **generic adaptive chosen-message attack** is an algorithm A using two oracles: a **hashing oracle** and a **signing oracle**. The input to A is a public key n.

Let H be a hash function. The **user's hashing oracle for** H prints $H(x)$ given x. The **user's signing oracle for** H is Algorithm S: given a message m, it prints a standard signature (r, h, f, s, t) for m under n, where r is a uniform random element of $\{0, 1, \ldots, 2^{256} - 1\}$ generated independently for each oracle query.

By definition A is **successful** if it prints a vector (r, m, h, f, s) such that

- r is in $\{0, 1, \ldots, 2^{256} - 1\}$,
- m is a message that was *not* a query to the signing oracle,
- (r, m) *was* a query to the hashing oracle, with output h,
- f is in $\{-2, -1, 1, 2\}$, and
- s is an integer such that $s^2 \equiv fh \pmod{n}$.

The **generic success probability of** A is the probability that A is successful for a random public key n, given the user's hashing and signing oracles for a uniform random hash function H.

How to compute square roots. Let A be a generic adaptive chosen-message attack. Here is an algorithm A' that, given an integer y coprime to n and known to be a square modulo n, attempts to compute a square root of y modulo n.

The idea of A' is to run A with oracles that simulate the behavior of Algorithm S with a uniform random hash function H. The oracles build a dictionary D mapping pairs (r, m) to scribbles (h, f, s, t), and a second dictionary E mapping pairs (r, m) to vectors (h, f, w). Initially both dictionaries are empty.

The **attacker's signing oracle** handles a query m as follows. Generate a uniform random element r of $\{0, 1, \ldots, 2^{256} - 1\}$. If (r, m) is in E, abort. If (r, m) is in D, print (r, h, f, s, t) where $(h, f, s, t) = D(r, m)$. Otherwise generate a scribble (h, f, s, t) with Algorithm V, set $D(r, m) \leftarrow (h, f, s, t)$, and print (r, h, f, s, t).

The **attacker's hashing oracle** handles a query (r, m) as follows. If (r, m) is in D, look up $(h, f, s, t) = D(r, m)$, and print h. If (r, m) is in E, look up $(h, f, w) = E(r, m)$, and print h. Otherwise generate a vector (h, f, w) with Algorithm W, set $E(r, m) \leftarrow (h, f, w)$, and print h.

If A succeeds then A' finds a square root of y as follows. Consider the vector (r, m, h, g, s) printed by A. By assumption (r, m) was a query to the hashing oracle, and m was not a query to the signing oracle, so (r, m) must be in E. By assumption the hashing oracle printed h given (r, m), so $E(r, m) = (h, f, w)$ for some f, w. Now $w^2 y \equiv fh$ with $f \in \{-2, -1, 1, 2\}$ by Theorem 5.5, and $s^2 \equiv gh$ with $g \in \{-2, -1, 1, 2\}$ by assumption. If h has a nontrivial factor in common with n then A' finds a square root of y as in Algorithm S. Otherwise the squares fh and gh must be equal, so $w^2 y \equiv s^2$, and w is coprime to n; so A' finds a square root of y by computing $w^{-1}s \bmod n$.

Theorem 6.1. *If A performs at most L hashing queries and at most M signing queries, then A' prints a square root of y with probability at least $\epsilon - 2^{-256}LM$ where ϵ is the generic success probability of A.*

Proof. When A is run with the user's oracles for a uniform random hash function H, the probability of a **collision**—i.e., of the signing oracle generating r such that

(r, m) was a previous hashing query—is at most $2^{-256}LM$. Therefore there is probability at least $\epsilon - 2^{-256}LM$ that A succeeds without collisions.

In the absence of collisions, the attacker's oracles produce any particular sequence of results with the same conditional probability as the user's oracles. Indeed, in this case, the attacker's signing oracle does not abort; the attacker's hashing oracles produce uniform random hash values by Theorem 5.5; the attacker's signing oracles produce uniform random hash values, and the corresponding scribbles, by Theorem 5.3 and Theorem 5.4.

Therefore the probability of A succeeding without collisions using the attacker's oracles is at least $\epsilon - 2^{-256}LM$. □

Notes. The idea of proving security against generic attacks was popularized by Bellare and Rogaway in [2]. Of course, such a proof says nothing about the security of the hash functions actually used in practice. Perhaps every easily computed hash function is insecure.

The method of proof in this section is similar to the method outlined in [4]. The main difference is that Algorithm S selects a particular square root of a, whereas [4, Theorem 6.1] requires that the signer select a uniform random square root of a.

References

[1] Victoria Ashby (editor), *First ACM conference on computer and communications security*, Association for Computing Machinery, New York, 1993.

[2] Mihir Bellare, Phillip Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, in [1] (1993), 62–73.

[3] Mihir Bellare, Phillip Rogaway, *The exact security of digital signatures: how to sign with RSA and Rabin*, in [15] (1996), 399–416.

[4] Mihir Bellare, Phillip Rogaway, *The exact security of digital signatures: how to sign with RSA and Rabin*, draft available as [3], newer draft available as `http://wwwcsif.cs.ucdavis.edu/~rogaway/papers/exactsigs.ps.gz`.

[5] Daniel J. Bernstein, *Floating-point arithmetic and message authentication*, submitted for publication; draft available as `http://cr.yp.to/papers/hash127.dvi`.

[6] Henri Cohen, *A course in computational algebraic number theory*, Springer-Verlag, Berlin, 1993.

[7] Ivan B. Damgård (editor), *Advances in cryptology—EUROCRYPT '90*, Lecture Notes in Computer Science 473, Springer-Verlag, Berlin, 1991. ISBN 3–540–53587–X.

[8] Richard A. DeMillo, David P. Dobkin, Anita K. Jones, Richard J. Lipton (editors), *Foundations of secure computation*, Academic Press, New York, 1978. ISBN 0-12-210350-5.

[9] Yvo Desmedt (editor), *Advances in cryptology—CRYPTO '94*, Lecture Notes in Computer Science 839, Springer-Verlag, Berlin, 1994.

[10] Louis Claude Guillou, Jean-Jacques Quisquater, *Precautions taken against various potential attacks in ISO/IEC DIS 9796*, in [7] (1991), 465–473.

[11] Burt Kaliski, Matthew Robshaw, *The secure use of RSA*, CryptoBytes **1.3** (1995), 7–13.

[12] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 3rd edition, Addison-Wesley, Reading, 1997. ISBN 0–201–89684–2.

[13] Arjen K. Lenstra, Hendrik W. Lenstra, Jr. (editors), *The development of the number field sieve*, Lecture Notes in Mathematics 1554, Springer-Verlag, Berlin, 1993.

[14] Hendrik W. Lenstra, Jr., *Factoring integers with elliptic curves*, Annals of Mathematics **126** (1987), 649–673.

[15] Ueli M. Maurer (editor), *Advances in cryptology—EUROCRYPT '96*, Lecture Notes in Computer Science 1070, Springer-Verlag, Berlin, 1996. ISBN 3–540–61186–X.

[16] Ralph Merkle, *A fast software one-way hash function*, Journal of Cryptology **3** (1990), 43–58.

[17] Preda Mihailescu, *Fast generation of provable primes using search in arithmetic progressions*, in [9] (1994), 282–293. MR 95j:11122.

[18] Carl Pomerance, John L. Selfridge, Samuel S. Wagstaff, Jr., *The pseudoprimes to* $25 \cdot 10^9$, Mathematics of Computation **35** (1980), 1003–1026.

[19] Michael O. Rabin, *Digitalized signatures*, in [8] (1978), 155–168.

[20] Michael O. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, Technical Report 212, MIT Laboratory for Computer Science, 1979.

[21] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), 120–126.

[22] Hugh C. Williams, *A modification of the RSA public-key encryption procedure*, IEEE Transactions on Information Theory **26** (1980), 726–729.

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: `djb@pobox.com`